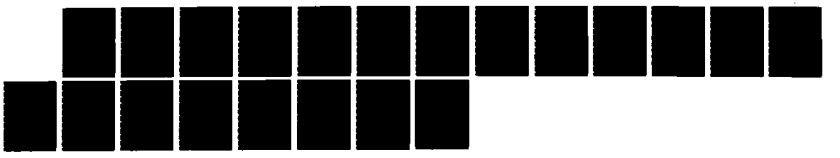
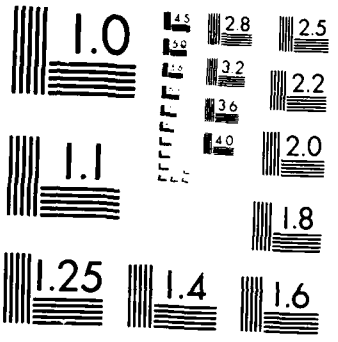
AD-A174 940 RESEARCH ON PROBLEM-SOLVING SYSTEMS(U) SRI INTERNATIONAL MENLO PARK CA D E WILKINS 13 OCT 86 AFOSR-TR-86-2190 F49620-85-K-0001 1/1

UNCLASSIFIED F/G 6/4 NL





Un
SECURIT

AD-A174 940

(2)

DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 86-2190
6a. NAME OF PERFORMING ORGANIZATION SRI International 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION AFOSR/NM
6c. ADDRESS (City, State and ZIP Code) 333 Ravenswood Ave Menlo Park, CA 94025	7b. ADDRESS (City, State and ZIP Code) Bldg 410 Bolling AFB DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR 8b. OFFICE SYMBOL (If applicable) NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F49620-85-K-0001
8c. ADDRESS (City, State and ZIP Code) Bldg 410 Bolling AFB DC 20332-6448	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: 61103F PROJECT NO.: 2304 TASK NO.: A7 WORK UNIT NO.:
11. TITLE (Include Security Classification) Research on Problem Solving Systems	
12. PERSONAL AUTHOR(S) Dr. David Wilkins	
13a. TYPE OF REPORT Final	13b. TIME COVERED FROM 851001 TO 860930
14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT 18
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

DTIC FILE COPY

DTIC ELECTE DEC 1 1986 E

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Sigillito	22b. TELEPHONE NUMBER (Include Area Code) (202) 767-5026 22c. OFFICE SYMBOL NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

SRI International

RESEARCH ON PROBLEM-SOLVING SYSTEMS

Annual Report
Covering Period October 1, 1985 to September 30, 1986

October 13, 1986

By: David E. Wilkins, Computer Scientist
Representation and Reasoning Group

Artificial Intelligence Center
Computer and Information Sciences Division

Prepared For:

Air Force Office of Scientific Research
Building 410
Bolling AFB, Washington, D.C. 20332
Attention: Dr. Vincent Sigillito

Contract No. F49620-85-K-0001
SRI Project 7898

APPROVED

Michael P. Georgeff, Program Director
Representation and Reasoning Group

Stanley J. Rosenschein, Director
Artificial Intelligence Center

SRI International

1 Introduction

Research on planning and problem-solving systems was begun at SRI International (SRI) in September 1979 under AFOSR sponsorship (SRI Project 8871, Contract No. F49620-79-C-0188). The current project (SRI Project 7898, Contract No. F49620-85-K-0001) continues this work. The research performed during previous years of the project is described in papers that appeared in the April 1984 issue of the *Artificial Intelligence Journal* [5] and the February 1985 issue of the *Computational Intelligence Journal* [7], as well as in previous annual reports. This report describes the research conducted during the past year.

Our main task under this program is to develop powerful methods of representing, generating, and executing hierarchical plans that contain parallel actions. Execution involves monitoring the state of the world and, possibly, replanning if things do not proceed as expected. Over the last few years, we have designed and implemented a system called SIPE (System for Interactive Planning and Execution Monitoring) [5], the purpose of which is to demonstrate the heuristic adequacy of our approach to this problem. Our basic approach is to work within the hierarchical-planning methodology, representing plans in procedural networks — as has been done in NOAH [4] and other systems. Several extensions of previous planning systems have been implemented, including the development of a perspicuous formalism for describing operators and objects, the use of constraints for the partial description of objects, the creation of mechanisms that permit concurrent exploration of alternative plans, the incorporation of heuristics for reasoning about resources, and implementation of a deductive capability.

Our research this year has concentrated on improving SIPE to provide a high-level reasoning capability for a mobile robot, such as the one currently being developed at SRI [3].

2 Using SIPE to Control a Mobile Robot

We have used SIPE at SRI to generate plans for a mobile robot [8]in an indoor environment consisting of hallways and offices. The problem with using such plans in the real world is the interface between the lower-level control routines and the planner. In the past year, we have designed this interface in more detail, and made improvements in SIPE in an effort to implement part of this design. Our interface design is described in detail in the enclosed technical note [8], and briefly summarized here.

The scope of our research does not include the development of lower-level control systems. Our goal is to investigate the utility of SIPE as the highest-level planning system in an integrated robot. Our concern is to develop a system that can work in the short term on simple but useful tasks in a constrained indoor environment, rather than an ultimate design for a completely robust, intelligent robot.

In particular, we do not want to commit ourselves to particular representations or designs for the lower-level systems. Instead, we are using a heuristic interface language that requires SIPE to plan to a suitably low level before executing actions, and has SIPE accept suitably low-level feedback from other systems. We want the interface language and its extensions to use concepts that can be accommodated by diverse low-level systems, so that we can take advantage of a diversity of research results, rather than commit ourselves to a single approach.

Our design for interfacing SIPE with low-level controllers involves treating them as programmable coroutines. There will be a bidirectional interface language for each controller that will allow the planner to instruct and program the controller, while the latter, in turn, will be able to inform and instruct the planner. The main idea is that the controllers will not be simply subroutines to be called, but will be full-fledged routines, running concurrently

with SIPE, that can both send and receive information and requests. Thus, a sensor can monitor the world continuously and alert the planner when a certain condition arises. The planner will be able to program such a sensor by telling it what conditions are expected and which unexpected conditions, if they occur, should generate interrupts.

During the past year we attempted to implement an interface with a simple sonar sensor. The language described in Section 4 defines our target interaction between the planner and the sonar monitor. Some effort was expended in studying the work of others to determine if a sonar monitor with these capabilities already existed. Our conclusion was that the problem of detecting walls, doors, and objects from sonar readings, while solvable, has not yet been solved in a robust manner. As described in Section 5, the systems we investigated were easily confused in a rich, real-world environment. For this reason, we have tested SIPE on our robot simulator, which simulates execution of actual robot motor commands and displays the resulting motion of the robot. SIPE is given (by the user) descriptions of unexpected world states, and correctly updates the plan being executed to react to these dynamic changes. This testing on the simulator has stimulated several extensions to the system, which are briefly described in the next section.

3 Summary of Accomplishments

Testing the system with the robot simulator was sufficiently challenging to produce a number of improvements in SIPE that will be necessary for controlling a mobile robot. In addition, several ideas that were not implemented were explored. These ideas and improvements are summarized here, with the most important ones described in more detail either in the enclosed technical note [8]or later in this document.

- The replanner, which initially operated automatically, can now by operated interactively as well. This is advantageous for debugging new applications and controlling the system.

- The replanning algorithm was improved as described in detail in section 6. Several ideas not implemented are also discussed.

- The ability to intermingle planning and execution within SIPE was implemented. This enabled the planner to produce an executable plan for our sample problem in 9 seconds instead of 35, as described in detail in [8].

- Last year, we identified a problem with hierarchical planning. During this past year, three different solutions to it were implemented in SIPE and compared, as described in detail in [8].

- SIPE now permits certain goals to be flagged as desirable for being accomplished by instantiation. In general this is not done, because choosing a wrong instantiation may prevent a solution from being found, but letting the user invoke this behavior when desired makes the system more flexible.

- SIPE now invests more effort in checking consequences of PRED constraints when matching two variables. This procedure results in more expensive matches, but also cuts down on the search space by finding incompatible constraints more quickly. This is discussed in more detail in Section 7.

- Many changes were made to improve efficiency, including checking for subsumption of constraints. Timing tests were done and substantial gains were realized by these changes. They are discussed in more detail in Section 8.

- SIPE's deductive capability was extended by allowing universal variables in the preconditions. This is a powerful extension and is described in detail in [8].

- An alternative implementation of SIPE was created that uses flavors (the Symbolics object-oriented programming package) instead of arrays as its basic representation. The flavor version was slower but may prove desirable in the future because of its modifiability.

- A graphics package was implemented for SIPE, so that plans can be viewed graphically on the screen. Graphics are useful for checking the correctness of plans that are produced and for debugging both the system and applications that are done in the system.

- Considerable effort was invested in studying the robotics and low-level perception work of others to determine an appropriate interface for SIPE. The results are described in Section 5.

4 Sonar Interface

Once SIPE has activated the sonar controller, the latter runs as a separate routine and may send information to the planner at any time. Such information usually includes the estimated location of the robot and some description of why the action being performed is not proceeding as expected. The description may include the robot's power being low, the robot being incapacitated for unknown reasons, an object being discovered with blocks the path, a door being closed, or failing to observe an object that was expected to be in view. Upon receipt of such information, SIPE will begin planning and may issue new instructions to the controller.

In our indoor environment, SIPE gives the following primitive commands to the sonar controller:

- GO FORWARD distance
  In this case the controller does not try to track along a wall. It simply moves forward the given distance. It may veer slightly one way or other to avoid an object.

- TURN theta
  This primitive simply turns the robot the specified amount while it is standing in place.

- GO TO NEXT DOOR theta side distance
  This command assumes you are in a hall, and tries to go in direction theta until it sees a door on the specified side. The distance the robot is expected to travel can be provided if the sonar controller can make use of this information.

- APPROACH NEXT OBJECT theta distance
  The robot should approach the nearest object in the direction theta, and proceed until the object is touching a bumper. This moves the robot slowly near the end of its travel and inhibits lower-level routines whose purpose is to avoid collisions. Again, the expected distance to the object can be provided.

- FOLLOW WALL theta distance
  The controller attempts to follow a wall (in the general given direction) down a hallway for a given distance. It may veer as far from side to side as is necessary to get around obstacles. This rather complex subsystem will involve special algorithms for obstacle avoidance. Work on such algorithms is ongoing at several places (e.g., [1,6]), and SIPE should be able to make use of developments in this area. SIPE only uses this command if, to the best of its knowledge, the hallway is passable. Otherwise, SIPE will plan to clear the hallway or to pursue another means of achieving its goals. Thus the obstacle-avoidance algorithm will often be addressing solvable problems.

- FOLLOW WALL TO DOOR n theta distance
  This is the same as FOLLOW WALL except that the controller is expected to recognize doorways as it proceeds down the hall, and stop at the *nth* one down the hall.

- GO THROUGH DOOR theta distance
  This is the most complicated primitive. It may not be reasonable to do this action with only sonar sensors, because the robot is nearly as wide as a standard door. Visual input may be needed to align the robot with the door. The robot should be in front of a door, which is in direction theta. This routine should center the robot in front of the door and go forward through the door, assuring clearance on both sides, until the center of the robot is the given distance through the door.

- FIND NEXT DOOR theta
  This primitive assumes that the robot is in a hall but may otherwise be lost. It attempts to go in direction theta until it finds a door on either side. It stops at the first door, and describes its expected location.
- FIND NEXT OBJECT theta (optional distance)
  This primitive tries to go in direction theta until its path is blocked by an object, and describes its expected location. This command is also useful when the robot is lost.

5 Low-Level Sonar Perception

During the past year, we investigated the possibility of implementing the above interface in a robust way so that our robot could be controlled in our building without special constraints on its environment. If we used only sonar sensors, this problem proved to be too difficult to solve with our resources. The sonar data are noisy, and we found it hard to determine which detected lines are walls, which are doors, and which are objects in the hallway. The robot can easily become confused when people walk by, when it turns corners, or when the hallway is cluttered.

Two approaches being developed concurrently here at SRI were investigated as possibilities for the sonar controller. One was the robot control system implemented in PRS [2]by Georgeff et al. While this system was used to control the robot, it made many limiting assumptions (e.g. all hallways are the same width, all corners are rectangular, all doors are open and unobstructed) and was easily confused (e.g. when people walked by). Once it became confused, it was not able to recover. We decided that more useful improvements could be added to the planner by testing it on the robot simulator.

The other approach being pursued at SRI is the situated-automata-based system of Rosenschein and Kaelbling [3]. Their system is proving to be robust and reliable at following hallways, but cannot yet count doors or go through them. This system appears to provide a

good basis for our envisioned sonar controller, but the implementation was not finished in time for our project to take advantage of it.

6 Improving the Replanning Algorithm

Below we discuss many ideas that were considered on how replanning should backtrack after the initial attempt at replanning fails. Problems that were encountered in the mobile robot domain motivated much of this endeavor. The conclusion we reach in all but the simplest cases is that the process of intelligently deciding where to replan is too expensive to implement. Thus, SIPE currently reinitializes the problem in this situation.

The following problem is a motivating example. Suppose the plan calls for walking down a hall and going through the first door to a conference room. If the robot discovers the first door is blocked, is it possible to replan to use the second door without redoing the part of the plan that comes after entering the conference room? If so, how do we decide which part of the plan has to be redone and which part can be kept intact? It is assumed that the planner knows which node at the most primitive level is causing a problem in the plan, called the failed node, which in the example might be a node requiring the conference room door to be unblocked.

Replanning may take place at any level in the hierarchy. A node in a hierarchical plan defines a *wedge* [7]. Starting with a node that was expanded by an operator application, a wedge of the plan is determined by following all its descendant links (in the current context) repeatedly (i.e. including descendants of descendants, and so on) to the lowest level. The node originally expanded by an operator application is called the *top* of the wedge. A wedge with its top at a high level in the hierarchy will generally contain many lower-level wedges within itself. To replan a node, SIPE simply replaces the wedge by the node at the top of

the wedge and calls the standard planner on the result [7].

6.1 Reinstantiation of Variables

One replanning possibility is to reinstantiate variables without changing anything else in the plan. For example, when getting screws from a hopper this procedure may be the correct response when you drop a screw – simply execute the same plan, returning to the hopper to pick up a different screw. However, the general problem is quite complicated. There are any number of constraints and instantiations on the plan variables from different parts of the plan. Reinstantiations involve removing some of these and trying to replace them. However, there are two problems: (1) it is not easy to determine all the consequences that have been propagated from the old instantiation choice (without implementing a truth maintenance system on top of the planner), and (2) in general, you must reinstantiate a whole subset of variables to solve the problem, not just one, and it is difficult to pick the correct subset out of the huge number of possibilities.

One idea we explored for choosing a set of variables used the following algorithm. Consider the variables in the failed node as possible candidates for reinstantiation. For each one, go up the hierarchy to the point where the variable was first introduced. This determines a wedge that in some sense is either causing or signalling the problem. Consider for reinstantiation only those variables whose instantiations were not forced by choices made inside this wedge. The intuition behind this approach is that because an instantiation was not forced by this wedge, the wedge itself might quite likely work without modification on another instantiation of the same variable. The check for where choices are forced is simple in SIPE, because all constraints (including instantiation constraints) are posted relative to choice points, and it is easy to determine which choice points are in a wedge.

Considering the above problems, and the fact that the reinstantiating process is not

necessarily that much cheaper than the planning, we did not implement any reinstantiation beyond that described in [7]. Instead, SIPE relies on popping up to some higher level in the hierarchical plan and replanning.

6.2 Replanning a Node

To replan a single node, SIPE simply replaces the wedge with the node at the top of the wedge and calls the standard planner on the result. This process effectively removes many actions from the plan at the most primitive level. However, there are two problems encountered in replanning just a single node rather than redoing the entire plan. First, the particular node must be selected; second, something must be done if the new plan produced causes problems in subsequent parts of the original plan. We can avoid these problems if we choose an ancestor of the failed node, high enough in the hierarchy, so that its wedge will contain everything that must be replanned, thus ensuring that we will produce a correct plan. In our example, we would like to back up to the goal of getting the robot in the conference room. Backing up to a more primitive goal than this one will not produce a correct plan, and backing up further will cause the system to replan more than is necessary.

6.3 Choosing a Wedge

The default solution, resolving the whole problem, is equivalent to backing up to the largest wedge with the highest root and replanning it. Replanning a node means that we choose a wedge that is not that high and replan it. However, we would like to pick a wedge that is high enough (if possible) to ensure success of the replanning. If we do not, we have to explore a whole search space of possible wedges, where each one may be as difficult as the original problem. Because it has proven to be difficult to select the proper wedge, SIPE has opted simply to redo the original plan in most cases rather than explore this search space.

One of the algorithms we considered for choosing a wedge entailed going back to the first node that had an untried operator, i.e. a node that represents a choice point at which the system has other (as yet untried) choices of which operator to apply. This algorithm does not give the desired results because the replanning often does not need to apply different actions. In the robot domain, it is frequently the case that the same actions must be done, but in the altered world state used for the replanning, these same actions will result in different instantiations for many variables. The ideal wedge for replanning often requires (in our test domains) applying the same operators that were used in the original plan for a level or two, but then applying different operators at lower levels. Therefore, applying the above-mentioned algorithm often results in selecting a wedge that is not high enough in the hierarchy, which dooms the replanning to failure.

Another attractive algorithm is to return to the node which first introduced the variables that are the arguments of the failed node. In our example, if the failed node involves the conference room door being blocked, then going up the hierarchy to a level at which the door is not mentioned does get us to the ideal replanning level, attempting to get the robot into the conference room. However, this algorithm often returns all the way to the top. It is a reasonable solution to the problem of choosing the wedge, and in fact was implemented in SIPE, but it does not often succeed because of considerations mentioned in the next subsection.

It is also possible to combine the above two algorithms to ensure a conservative choice of wedges (in the sense of assuring success in the replanning). Such an algorithm would return to a higher level of the hierarchy at which all the variables of the failed node disappear, and then continue up until it finds a node with an untried choice.

6.4 Replanning Causing New Problems

Replanning at one point may cause problems in the remaining part of the plan (that we hope to reuse). SIPE can detect such problems during the replanning process. Problems may include either phantoms or preconditions that are no longer true. If reusing some part of the old plan were a high priority, we could overcome this problem by changing the phantoms to goals, replacing the wedge determined by a precondition (i.e. follow the precondition up to the level where it was first introduced) with the wedge's root, and calling the normal planner again. SIPE does not do this; rather, it simply retries the original problem if there are any problems during the replanning. We chose this course because (1) constantly checking for these problems and correcting them is time consuming, (2) the occurrence of problems means less of the old plan is being reused, and (3) there is no guarantee that the plan we are attempting to reuse is part of a valid solution, while starting over guarantees a solution will be found if one exists.

It might be possible to make use of previous planning when starting over on the problem. The following idea was explored but not implemented. One could record the operator application choices that were made during the original successful planning session and use these to make operator choices on the second attempt. However, we have not determined how to record these decisions, or how to make use of this record once something slightly different has occurred in the planning. We recognize that the correct operator choice is often determined by different instantiations of variables to which this scheme is not sensitive.

7 Matching Variables with PRED Constraints

PRED constraints in SIPE give a disjunction of sets of instantiations, any one of which can be used to assure that a particular predicate is made true. When matching two variables, the

second of which has a PRED constraint, the system must determine if a set of instantiations in the PRED constraint will be compatible after these two variables are matched. SIPE used to simply take the member of the set that corresponds to the first variable, and check whether it was an acceptable match with the first variable. This was efficient, but permitted invalid matches (because other members of the set may have incompatible constraints), which would later cause the search to backtrack past this point after exploring a blind alley. The problem with checking every member of the set is that this would propagate matches throughout the system (when PRED constraints in this set are matched recursively). In the worst case, the system might run through all the constraints each time it matched a variable. This is computationally unacceptable. Another problem is that loops may be created because the system may recursively match the two original variables again.

In the mobile robot problem, invalid matches, in addition to allowing the searching of blind alleys, caused problems in the replanner. The matching algorithm was extended to check every element in a set of instantiations from a PRED constraint. To avoid the above problems, this extensive match is only done at the top level of recursive calls to the matcher. At lower levels of the recursion, the system still goes through the whole set, but this time only checks the variables that are instantiated (effectively assuming the uninstantiated variables will be acceptable). This avoids both the problem of needing to check for loops and of the matching becoming prohibitively expensive.

This more expensive matching algorithm proved to be very worthwhile in the mobile robot domain, because the rejected matches constrain the search space and minimize problems with replanning. The idea of checking the instantiated variables at the lower levels of recursion (instead of simply checking one variable, as was done initially) is critical for achieving our level of performance,because it is frequently the case that most variables in a problem are instantiated.

8 Efficiency considerations

Several changes were made to improve the efficiency of the system. The improvements were significant, as shown in Table 1. The primary change was checking for PRED constraints that are subsumed by other PRED constraints (i.e. they are already implied by other constraints). Such subsumed constraints are generated frequently by the system as it continually matches conditions during the planning process. The matching process can be speeded up considerably by either not posting subsumed constraints or not retrieving them, because fewer constraints need to be checked. The goal is to test subsumption in a computationally inexpensive way.

In addition to a disjunction of sets of possibilities, a PRED constraint contains the predicate for which it was originally generated. The subsumption-checking algorithm implemented during the past year makes use of this as follows. Given two PRED constraints on the same variable, the two original predicates are matched against each other. If their arguments match exactly or match with universal variables (i.e. without forcing any other instantiations), then only the more recent PRED constraint need be retrieved, because its set of possibilities will be included in the other constraint. Note that both constraints must still be posted since backtracking may eliminate the more recent one while keeping the older one intact. This subsumption check is inexpensive at retrieval time because an exact match is required (one never needs to match two uninstantiated variables).

When posting PRED constraints, SIPE now checks for exact matches against previous PRED constraints (as above), and then compares the two disjunctions. If the constraint to be added does not further constrain the disjunction, then it can be ignored and not posted at all. In addition, the system now orders the constraints it retrieves, to facilitate efficient matching.

---

[1] Run time is given in arbitrary units.

Function	Before Improvements		After Improvements	
	Run Time Inclusive[1]	Calls Exclusive	Run Time Inclusive	Calls Exclusive
Overall search	195	7	95	7
Matching a condition	152.4	340	76	340
Matching two variables	151.9	78,435	67	44,175
Deducing effects of a node	127	98	67	97
Solving global constraints	10.6	4	3.3	4

Table 1: Efficiency Improvements in SIPE

The performance improvements affected by the changes made here, and others too detailed to describe, are shown in Table 1. More than three-quarters of the total computation was originally being used to match two variables, and most of this was done during the process of deducing effects of a node. The improvements outlined above enabled us to reduce the total effort and the effort spent on matching variables by more than a factor of two. The effort spent in solving the global constraint satisfaction problem was reduced by a factor of three. The number of times that two variables were matched was cut by more than 40 percent, primarily because of the check for subsumed PRED constraints.

9 Publications and Talks

During the past year, a paper [9]by David Wilkins describing problems with hierarchical planning was published in the *Proceedings of the European Conference on Artificial Intelligence*, and delivered to a large audience at the conference. A paper describing the application of SIPE to the mobile robot domain was written and accepted for a special issue of the *Journal of Man-Machine Systems*. (This paper is also being distributed as an SRI Technical Note

and is enclosed with this report.) David Wilkins participated in the Workshop on Planning and Reasoning about Action at Timberline, Oregon, in June. This workshop included fruitful discussion among several of the world's top planning researchers.

References

[1] Brooks, R., "A Subdivision Algorithm in Configuration Space for Findpath with Rotation", *Proceedings IJCAI-83*, Karlsruhe, Germany, 1983, pp. 799-806.

[2] Georgeff, M., Lansky, A. and Schoppers M., "Reasoning and Planning in Dynamic Domains: An Experiment with a Mobile Robot", Technical Note 380, SRI International Artificial Intelligence Center, Menlo Park, California, 1986.

[3] Rosenschein, S., Fischler, M., and Kaelbling, L., "Research on Intelligent Mobile Robots", Final Report, Project 7390, SRI International Artificial Intelligence Center, Menlo Park, California, 1986.

[4] Sacerdoti, E., *A Structure for Plans and Behavior*, Elsevier, North-Holland, New York, 1977.

[5] Thorpe, C., "Path Relaxation: Path Planning for a Mobile Robot", *Proceedings AAAI-84*, Austin, Texas, 1984, pp. 318-321.

[6] Wilkins, D., "Domain-independent Planning: Representation and Plan Generation", *Artificial Intelligence 22*, April 1984, pp. 269-301.

[7] Wilkins, D., "Recovering from Execution Errors in SIPE", *Computation Intelligence 1*, February 1985, pp. 33-45.

[8] Wilkins, D., "High-Level Planning in a Mobile Robot Domain", Technical Note 388, SRI International Artificial Intelligence Center, Menlo Park, California, 1986.

[9] Wilkins, D., "Hierarchical Planning: Definition and Implementation", *Proceedings of the Seventh ECAI*, Brighton, England, 1986, pp. 466-478.

END

4-87

DTIC